

Documento: ISS4U_Sviluppo-app

Delivery: 04

SVILUPPO APPLICAZIONE

Documento per lo sviluppo finale dell'applicazione

Andrea Gravili, Maria Laura La Face, Matteo Parma

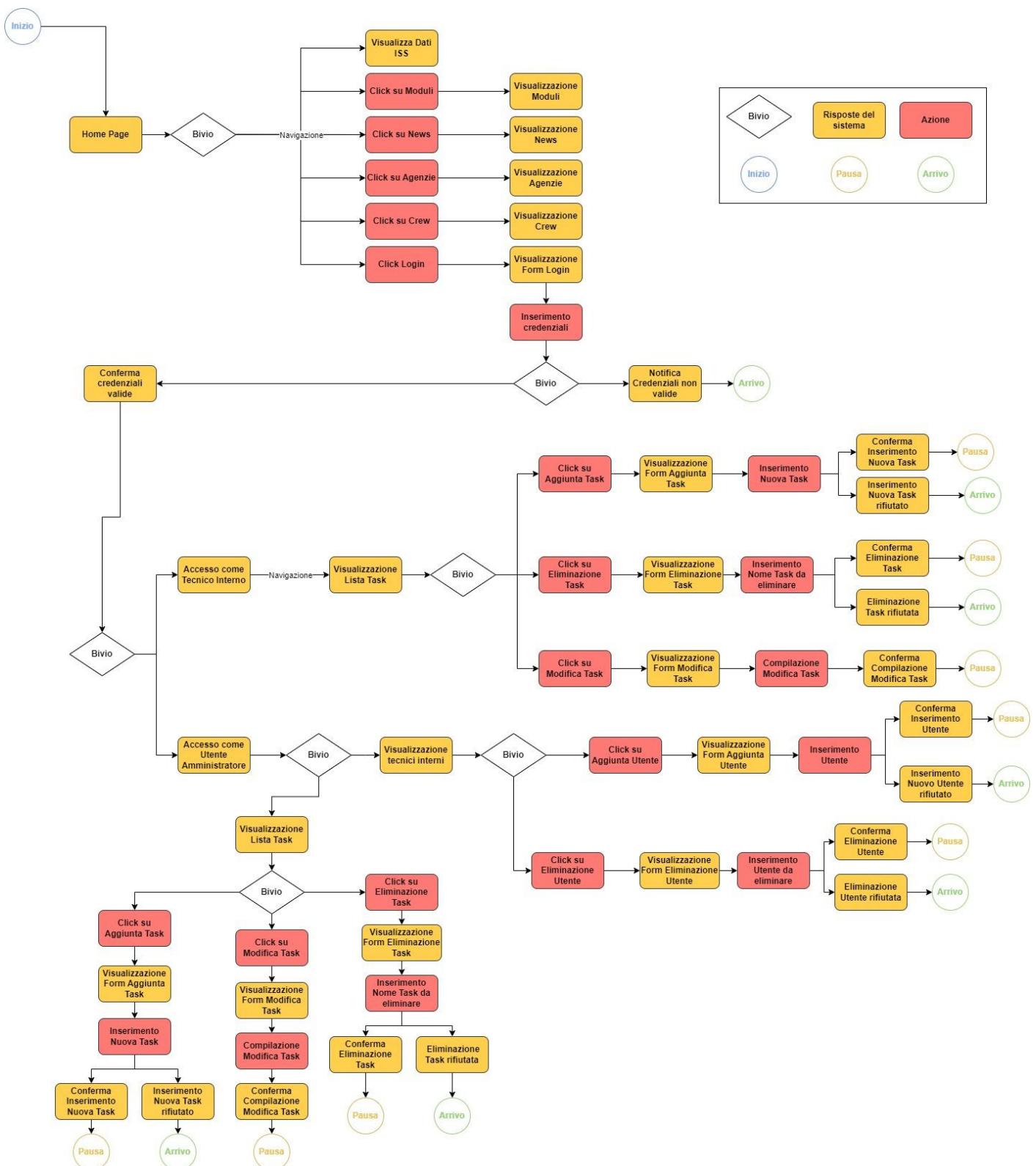
Scopo del documento.....	3
User Flow.....	3
Application Implementation and Documentation.....	5
Project Structure.....	5
Project Dependencies.....	7
Project Data or DB.....	7
Project APIs.....	11
Resource Extraction from Class Diagram.....	11
Resource Models.....	12
Sviluppo APIs.....	12
Lista task.....	13
Selezione task.....	14
Aggiunta task.....	15
Eliminazione task.....	17
Modifica task.....	18
Lista moduli.....	19
Selezione modulo.....	20
Lista crew.....	20
Lista agenzie.....	21
Lista news.....	21
Lista missioni.....	21
Login.....	22
Aggiunta tecnico interno.....	23
Eliminazione tecnico interno.....	24
API Documentation.....	25
FrontEnd Implementation.....	26
Home page - componenti.....	27
Agenzie.....	28
Astronauti.....	29
History mission.....	30
News.....	31
Login.....	31
Visualizza task.....	32
Aggiunta task.....	33
Eliminazione task.....	34
Modifica task.....	35
Aggiunta utente.....	36
Elimina utente.....	37
GitHub Repository & Deployment Info.....	38
Testing.....	38

Scopo del documento

Il presente documento riporta tutte le informazioni necessarie per lo sviluppo di una parte dell'applicazione ISS4U. In particolare, presenta tutti gli artefatti necessari per realizzare i servizi dell'applicazione, tra cui quelli per la gestione dei tecnici interni e delle loro task, oltre ad altri servizi utilizzabili dall'utente generico.

User Flow

In questa sezione del documento di sviluppo viene riportato lo user flow, che descrive il percorso che l'utente compie nel nostro sito ISS4U. L'utente generico può in ogni momento consultare la Home Page, la lista di astronauti e delle agenzie, le history mission e le news. Una volta fatto il login, il tecnico interno può visualizzare la lista delle sue task, modificarle e aggiungerle. Infine, l'utente amministratore può visualizzare la lista di tecnici interni, aggiungere utenti e assegnare task a questi ultimi.



Application Implementation and Documentation

Nelle sezioni precedenti abbiamo identificato le varie features che devono essere implementate per la nostra applicazione con un'idea di come il nostro utente finale può utilizzarle nel suo flusso applicativo. L'applicazione è stata sviluppata utilizzando NodeJS e VueJS. Per la gestione dei dati abbiamo utilizzato MongoDB.

Project Structure

Il progetto è stato diviso in due moduli, front-end e back-end, ognuno dei quali è organizzato in diverse cartelle.

- **Back-end:** Il tutto parte dal file `server.js` che si occupa di chiamare le rotte delle api invocate dal front-end. Queste rotte sono salvate nella cartella `routes`. Da queste rotte poi vengono invocate le funzioni necessarie a soddisfare le richieste. Queste funzioni si trovano in diversi file nella cartella `controllers`. I modelli usati per rappresentare i diversi dati si trovano in diversi file nella cartella `models`. Nella cartella `middlewares` sono presenti i file per la gestione dell'autenticazione e dell'autorizzazione. Nella cartella `test` si trovano tutti i casi di testing delle API. Sono presenti anche un `Procfile` per la configurazione di Heroku ed un file `swagger.json` per le api-docs
- **Front-end:** Composta dalle cartelle `api`, `assets`, `components` e `views` che sono contenute nella cartella `src`. Il front-end parte dal file `main.js` che crea e costruisce l'applicazione partendo dal file `App.vue`, con all'interno la struttura principale e le routes per le varie pagine visualizzabili. Da qui vengono create le pagine che si trovano all'interno della cartella `views` contenente le pagine principali visualizzabili dall'utente, all'interno ci sono frammenti di pagine che possono essere riutilizzate in futuro e implementate da pagine all'interno dei components. Ad esempio la pagina di login visualizzerà sia il componente della schermata home, la componente del login che permette l'autenticazione e il footer/header. All'interno di `assets` troviamo tutti i file che sono principalmente immagini che servono da supporto grafico alle pagine / componenti. Nella cartella `api` invece troveremo le api che inviano le richieste al back-end.

Nella seguente figura uno screenshot delle due strutture.

Back-End	Front-End
<pre>> controllers > coverage > middlewares > models > node_modules > routes > testing > views ⚙ .env ⚡ .gitignore {} package-lock.json {} package.json 💾 Procfile ⓘ README.md JS server.js {} swagger.json</pre>	<pre>> .vscode > dist > node_modules > public ▽ src > api > assets > components > views ⚙ .env 🌱 App.vue JS main.js JS Router.js ⚡ .gitignore ⚡ babel.config.js {} jsconfig.json {} package-lock.json {} package.json ⓘ README.md 📅 start.bat JS vue.config.js ⚡ webpack.config.js</pre>

Project Dependencies

I seguenti moduli Node sono stati utilizzati e aggiunti al file Package.Json

- express
- json
- mongoose
- jsonwebtoken
- bcryptjs
- cors
- dotenv
- multer
- node-fetch
- path
- moment
- swagger-ui-express
- jest
- supertest

Project Data or DB

Per la gestione dei dati utili all'applicazione abbiamo definito le seguenti strutture dati

Collection Name	Documents	Logical Data Size	Avg Document Size	Storage Size	Indexes	Index Size	Avg Index Size
missiones	5	1.91KB	392B	36KB	1	36KB	36KB
modulos	4	2.19KB	562B	36KB	1	36KB	36KB
news	3	1.71KB	584B	36KB	1	36KB	36KB
roles	2	113B	57B	36KB	1	36KB	36KB
tasks	5	1.86KB	382B	36KB	1	36KB	36KB
utentes	4	2.88KB	739B	36KB	1	36KB	36KB

Ognuna delle quali ha la seguente struttura dati:

Documento: ISS4U_Sviluppo-app

Delivery: 04

Mis^sione

```
_id: ObjectId('64e733f3164ea4bbaf0b489a')
titolo: "Expedition 1"
descrizione: "La prima Expedition segnò l'occupazione continua dell'ISS da parte del..."
immagine: "https://upload.wikimedia.org/wikipedia/commons/thumb/a/ab/ISS-Expediti..."
__v: 0
```

News

```
_id: ObjectId('64e72bf0d7b43769de2ebde5')
titolo: "Cardiac, Digestion Research Ahead of Space Delivery and New Crew"
descrizione: "A cargo craft is orbiting Earth today heading toward the International..."
link: "https://blogs.nasa.gov/spacestation/2023/08/23/cardiac-digestion-resea..."
copertina: "https://blogs.nasa.gov/spacestation/wp-content/uploads/sites/240/2023/..."
```

Modulo

```
_id: ObjectId('63d3973f44e3845fd02fc501')
nome: "Unity"
descrizione: "Il modulo Unity è stato il primo ad essere collegato alla ISS. Serve c..."
nazione: "Stati Uniti"
application: "Abitazione e supporto tecnico"
operator: "NASA"
contractors: "Boeing"
power: "8 kW"
mass: "19.7 t"
other_details: "Il modulo Unity è stato lanciato il 4 dicembre 1998 a bordo del shuttle Discovery"
launch_date: "4 dicembre 1998"
image: "https://img-new.cgtrader.com/items/1969625/48c1921ab3/unity-node-1-iss..."
```

Role

Ci sono solo due istanze di questa collezione: tecnico interno e amministratore, i loro id saranno poi utilizzati per definire l'attributo ruolo nelle istanze “utente”.

```
_id: ObjectId('63d3fff1e1a0280d07922cb7')
name: "tecnico_interno"
__v: 0
```

```
_id: ObjectId('63d3fff1e1a0280d07922cb8')
name: "amministratore"
__v: 0
```

Task

Ogni task ha come attributo *userId* l'id dell'utente associato a quella task

```
_id: ObjectId('64e64d390b820f21a34b1fe6')
nome: "Manutenzione Modulo"
data_inizio: 2023-09-11T00:00:00.000+00:00
data_fine: 2023-09-16T00:00:00.000+00:00
modulo: "Unity"
descrizione: "Verifica e sostituzione di componenti hardware e software all'interno ..."
completata: true
userId: ObjectId('63eb60a8b84d4069cf623747')
completata: false
```

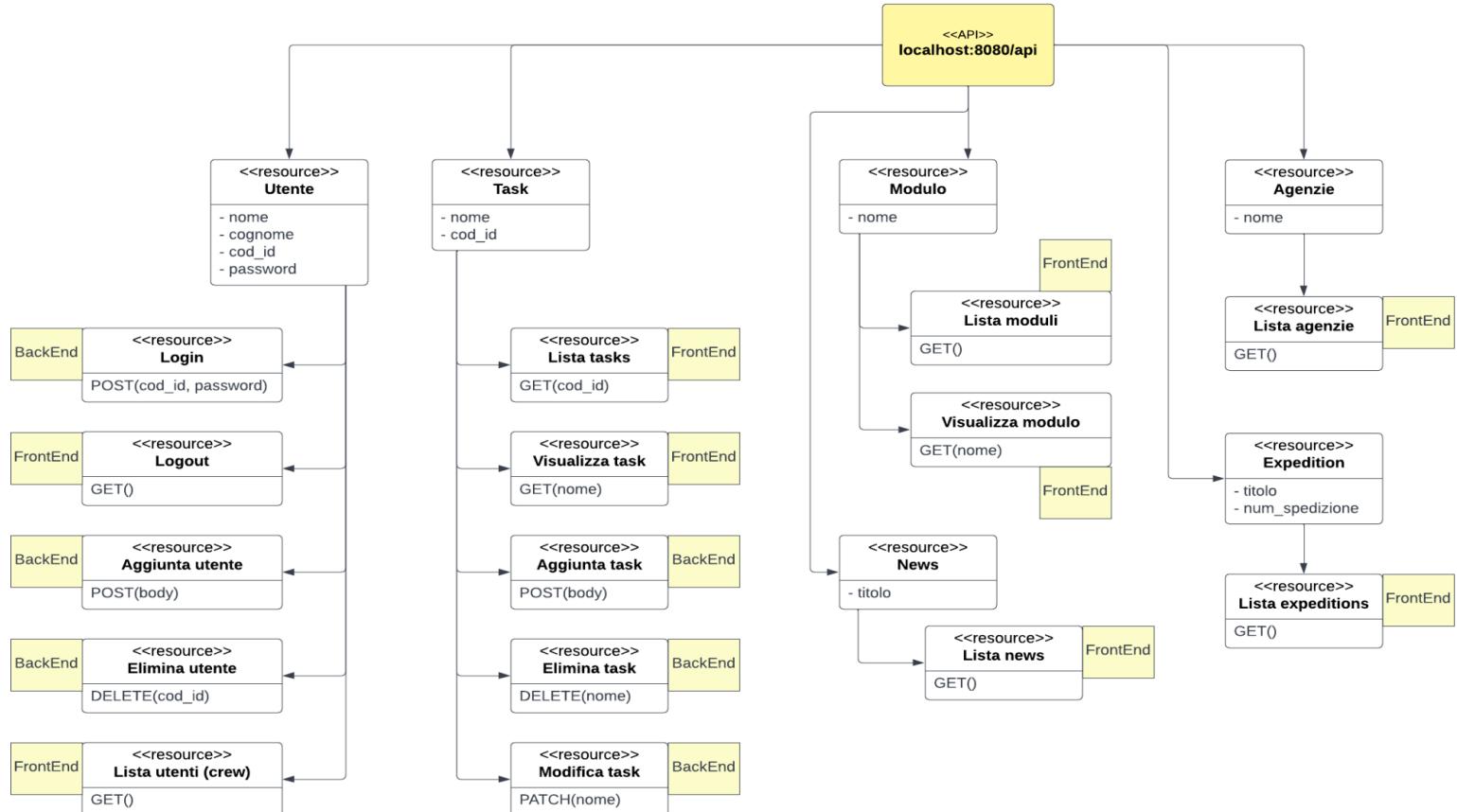
Utente

Ogni utente ha come attributo *role* l'id dell'istanza Role associata a quell'utente.

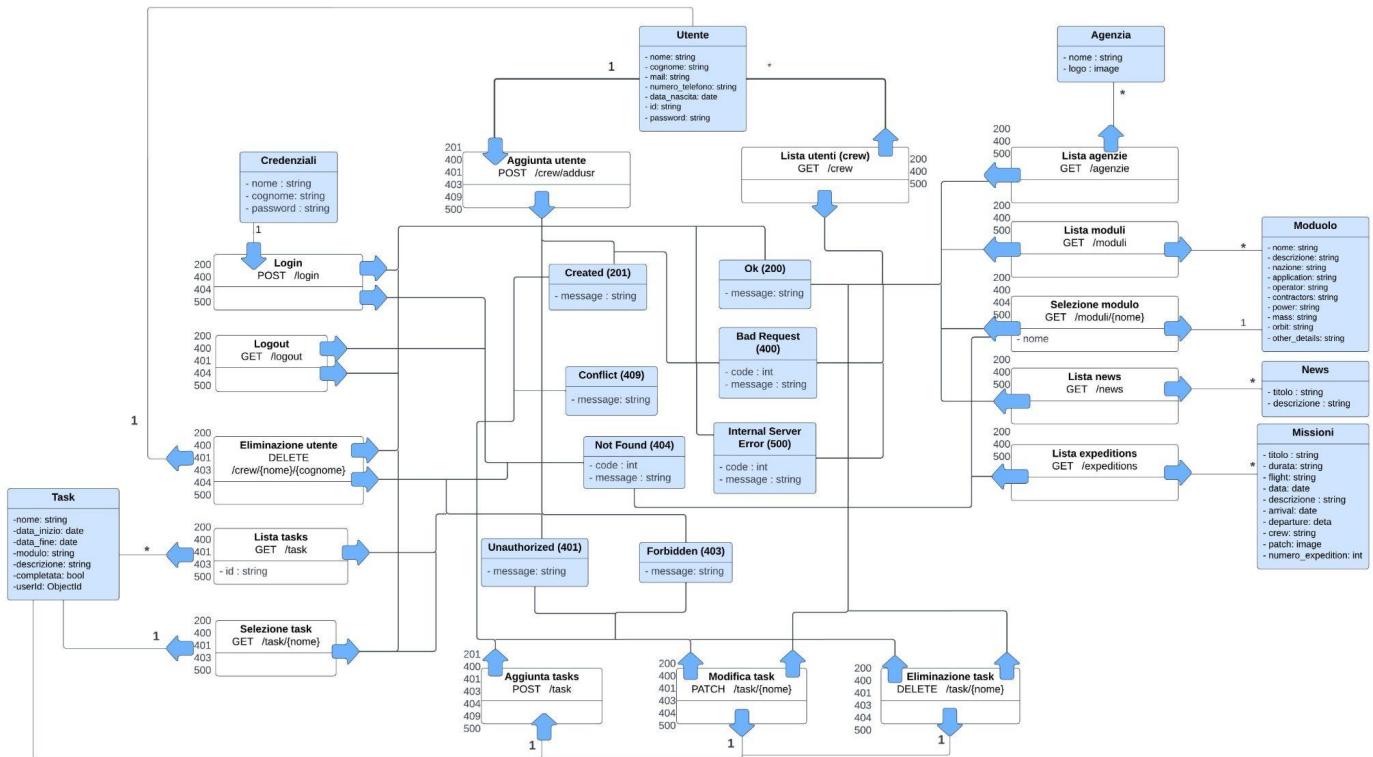
```
_id: ObjectId('63ea75a500109772ce17d0ed')
nome: "Samantha"
cognome: "Cristoforetti"
email: "samanta.cristoforetti@mail.com"
numero_telefono: "0123456789"
data_nascita: "26-04-1977"
password: "$2a$08$PZchgFlgWPndbV3RbtXrueRn6uRf29vvzfXKx6ByUvWcVRyzJMzU0"
__v: 0
role: ObjectId('63d3fff1e1a0280d07922cb8')
bio: "Samantha Cristoforetti è un'astronauta e aviatrice italiana, prima don..."
occupazione: "comandante"
immagine: "https://upload.wikimedia.org/wikipedia/commons/thumb/5/5e/Samantha_Cri..."
missioni: "Expedition 42/43 - Missione Minerva - Expedition 64"
```

Project APIs

Resource Extraction from Class Diagram



Resource Models



Sviluppo APIs

Le APIs sviluppate fanno riferimento alle:

- **Task**
 - Queste APIs sono accessibili solo agli utenti registrati, un utente anonimo riceverà un codice di stato 401 per ogni funzionalità
 - Lista Tasks
 - Selezione Task
 - Aggiunta Task
 - Eliminazione Task
 - Modifica Task
- **Moduli**
 - Lista Moduli
 - Selezione Modulo
- **Crew**
 - Lista Crew

- Agenzie
 - Lista Agenzie
- News
 - Lista News
- Missioni
 - Lista missioni

Dopo di che ci sono le APIs per la gestione dei tecnici interni e autenticazione. Le ultime due sono accessibili solo all'amministratore.

- Login
- Aggiunta tecnico interno
- Eliminazione tecnico interno

Lista task

Questa API restituisce l'elenco delle tasks salvate nel db utilizzando il metodo find({}). Le sue funzionalità variano in base al tipo di livello di autorizzazione dell'utente:

- Amministratore: il server restituisce l'elenco delle tasks di tutti i tecnici interni
- Tecnico interno: il server restituisce l'elenco delle tasks riferite a quell'utente

```
const getList= (req, res) => {
    Utente.findById(req.userId).exec((err, user) => {
        if (err) {
            res.status(500).send({ message: err });
            return;
        }

        //cerco il role dell'utente
        Role.findOne({_id: {$in : user.role}}, (err, role) => {
            if(err){
                res.status(500).send({ message: err });
                return;
            }

            if(role.name == 'amministratore'){
                //se amministratore prende tutte le task nel db
                Task.find({}, (err, data)=>{
                    if (err){
                        return res.json({Error: err});
                    }
                    return res.status(200).json(data);
                });
            }
        });
    });
}
```

```
    }else if(role.name == 'tecnico_interno'){
        //se tecnico interno prende solo le task riferite a quell'utente
        Task.find({userId: req.userId}, (err, data)=>{
            if (err){
                return res.json({Error: err});
            }
            return res.status(200).json(data);
        });
    });
}
});
```

Selezione task

Questa API ritorna la task con lo stesso nome che è stato passato come path parameter utilizzando il metodo findOne({}).

Se l'utente è un tecnico interno e la task richiesta non gli appartiene invia un codice di stato 403.

```
const getTask= (req, res) => {
    Utente.findById(req.userId, (err, user)=>{
        if(err){
            res.status(500).send({message: err});
            return;
        }
        //cerco il role dell'utente
        Role.findOne({_id: {$in : user.role}}, (err, role) => {
            if(err){
                res.status(500).send({ message: err });
                return;
            }

            Task.findOne({nome:req.params.nome}, (err, data) => {
                if(err){
                    return res.status(500).send({message: err});
                }
                if (!data){
                    return res.status(404).json({message: "Task doesn't exist."});
                }else{
```

```
//se amministratore oppure se tecnico interno e la task che sta  
cerca è la sua allora la restituisco  
        if ((role.name=='tecnico_interno' && data.userId==req.userId) ||  
role.name=='amministratore') {  
            return res.status(200).json(data);  
        } else {  
            return res.status(403).json({message: "La task non appartiene  
all'utente " + req.userId});  
        }  
    })  
});  
});  
};
```

Aggiunta task

Anche questa API, che prende come body parameter le informazioni riguardanti la task da aggiungere, si comporta diversamente in base al tipo di utente, infatti al tecnico interno la task sarà associata automaticamente a lui, mentre l'amministratore deve aggiungere il nome e il cognome del tecnico interno a cui associare la nuova task.

Un tecnico interno può associare la task solo a se stesso, se anche inserisse il nome e cognome di un altro utente questi verrebbero ignorati.

La task viene salvata nel db utilizzando il metodo save().

Il ruolo dell'utente viene trovato utilizzando i metodi findById() per trovare l'utente e finOne({}) per trovare il suo ruolo associato.

```
const addTask= (req, res) => {  
    Task.findOne({nome: req.body.nome}, (err, data) => {  
        //se non è già presente nel db  
        if(!data){  
            Utente.findById(req.userId).exec((err, user) => {  
                if (err) {  
                    res.status(500).send({ message: err });  
                    return;  
                }  
                //cerco il role dell'utente che ha inviato la richiesta  
                Role.findOne({_id: {$in : user.role}}), (err, role) => {  
                    if(err){  
                        res.status(500).send({ message: err });  
                        return;  
                    }  
                    //aggiungo la task al role  
                    role.tasks.push(task);  
                    role.save();  
                    res.status(201).json(task);  
                };  
            });  
        }  
    });  
};
```

```
        }
        if(role.name === "tecnico_interno"){
            const newTask= new Task({
                data_inizio: req.body.data_inizio,
                data_fine: req.body.data_fine,
                nome: req.body.nome,
                modulo: req.body.modulo,
                descrizione: req.body.descrizione,
                completata: req.body.completata,
                userId: req.userId //se un tecnico interno l'userId della task è l'Id dell'utente
            });
            newTask.save((err, data)=>{
                if(err) return res.status(500).json({Error: err});
                return res.status(201).json(data);
            });
        }else if(role.name==='amministratore'){
            //cerco il tecnico interno a cui assegnare la task
            Utente.findOne({nome: req.body.nomeuser, cognome: req.body.cognomeuser},
            (err, utente) => {
                if(err){
                    res.status(500).send({ message: err });
                    return;
                }
                if(!utente){
                    return res.status(404).json({message: "Utente non trovato"});
                }
                const newTask= new Task({
                    data_inizio: req.body.data_inizio,
                    data_fine: req.body.data_fine,
                    nome: req.body.nome,
                    modulo: req.body.modulo,
                    descrizione: req.body.descrizione,
                    completata: req.body.completata,
                    userId: utente._id //se amministratore lo userId della task è l'id del tecnico interno cercato
                });
                newTask.save((err, data)=>{
                    if(err) return res.status(500).json({Error: err});
                    return res.status(201).json(data);
                });
            });
        });
    });
} else{
    if(err) return res.status(500).json("Qualcosa è andato storto, riprova. ${err}");
}
```

```
        return res.status(409).json({message: "Esiste già una task con questo nome"})
    }
  });
};


```

Eliminazione task

La task con il nome passato come path parameter, viene eliminata dal db tramite il metodo `findOneAndDelete({})`. Un tecnico interno può eliminare solo una task a cui è associato.

```
const deleteTask= (req, res) => {
  Utente.findById(req.userId, (err, user)=>{
    if(err){
      res.status(500).send({message: err});
      return;
    }
    //cerco il role dell'utente
    Role.findOne({_id: {$in : user.role}}, (err, role) => {
      if(err){
        res.status(500).send({ message: err });
        return;
      }
      if(role.name=='amministratore'){
        //se amministratore elimino la task se esiste
        Task.findOneAndDelete({nome:req.params.nome}, (err, data) => {
          if(err){
            return res.status(500).send({message: err});
          }
          if(!data) {
            return res.status(404).json({message: "La task non esiste."});
          }else{
            return res.status(200).json(data);
          }
        });
      }else if(role.name=='tecnico_interno'){
        //se tecnico interno elimino la task se esiste e appartiene all'utente
        Task.findOneAndDelete({nome:req.params.nome, userId: req.userId}, (err,
data) => {
          if(err){
            return res.status(500).send({message: err});
          }
        })
      }
    });
  });
};


```

```
        if(!data) {
            return res.status(404).json({message: "La task non esiste o non
appartiene a questo utente."});
        }else{
            return res.status(200).json(data);
        }
    });
}
});
```

Modifica task

La task da modificare, tramite il metodo `findOneAndUpdate({})`, viene selezionata tramite il nome passato come path parameter e come body parameter i nuovi valori da aggiornare. Come per le precedenti APIs un tecnico interno può modificare solo le tasks a lui associate.

```
const modificaTask= (req, res) =>{
    Utente.findById(req.userId, (err, user) => {
        if(err){
            res.status(500).send({message: err});
            return;
        }
        //cerco il role dell'utente
        Role.findOne({_id: {$in : user.role}}, (err, role) => {
            if(err){
                res.status(500).send({ message: err });
                return;
            }
            if(role.name=='amministratore'){
                Task.findOneAndUpdate({nome: req.params.nome},
                {
                    data_inizio: req.body.data_inizio,
                    data_fine: req.body.data_fine,
                    nome: req.body.nome,
                    modulo: req.body.modulo,
                    descrizione: req.body.descrizione,
                    completata: req.body.completata
                },
                (err, data) => {
                    if(err){
                        res.status(500).send({ message: err });
                    }
                })
            }
        })
    })
}
```

```
        return;
    }
    if(!data){
        return res.status(404).json({message: "La task non esiste"})
    }else{
        return res.status(200).send({message: "Task modificata!"})
    }
});

else if(role.name=='tecnico_interno'){
    Task.findOneAndUpdate({nome: req.params.nome, userId: req.userId},
    {
        data_inizio: req.body.data_inizio,
        data_fine: req.body.data_fine,
        nome: req.body.nome,
        modulo: req.body.modulo,
        descrizione: req.body.descrizione,
        completata: req.body.completata
    },
    (err, data) => {
        if(err){
            res.status(500).send({ message: err });
            return;
        }
        if(!data){
            return res.status(404).json({message: "La task non esiste o non appartiene a questo utente"})
        }else{
            return res.status(200).send({message: "Task modificata!"})
        }
    });
}
);
};

};
```

Lista moduli

Restituisce l'elenco dei moduli salvati nel db utilizzando il metodo find({})

```
const getList= (req, res) => {
    Modulo.find({}, (err, data)=>{
        if (err){
```

```
        return res.status(500).json({Error: err});  
    }  
    if(!data) {  
        return res.status(404);  
    }else{  
        return res.status(200).json(data);  
    }  
})  
};
```

Selezione modulo

Questa API restituisce il modulo passando il nome come path parameter utilizzando il metodo findOne({})

```
const getModulo= (req, res) => {  
    let name= req.params.nome;  
    Modulo.findOne({nome:name}, (err, data) => {  
        if(err){  
            return res.status(500);  
        }  
        if (!data){  
            return res.status(404).json({message: "Module " + name + " doesn't exist."});  
        }else{  
            return res.status(200).json(data);  
        }  
    })  
};
```

Lista crew

Restituisce l'elenco della crew, utilizza il metodo find({})

```
const getList= (req, res) => {  
    Utente.find({}, (err, data)=>{  
        if (err){  
            return res.status(500).json({Error: err});  
        }  
        return res.status(200).json(data);  
    })  
};
```

Lista agenzie

Restituisce l'elenco delle agenzie, utilizza il metodo find({})

```
const getList= (req, res) => {
    Agenzia.find({}, (err, data)=>{
        if (err){
            return res.status(500).json({Error: err});
        }
        return res.status(200).json(data);
    })
};
```

Lista news

Restituisce l'elenco delle agenzie, utilizza il metodo find({})

```
const getList= (req, res) => {
    News.find({}, (err, data)=>{
        if (err){
            return res.status(500).json({Error: err});
        }
        return res.status(200).json(data);
    })
};
```

Lista missioni

Restituisce l'elenco delle missioni, utilizza il metodo find({})

```
const getList= (req, res) => {
    Missione.find({}, (err, data)=>{
        if (err){
            return res.status(500).json({Error: err});
        }
        return res.status(200).json(data);
    })
};
```

Login

Passando come body parameter il nome, il cognome e la password permette di autenticarsi, come tecnico interno o amministratore, e ad accedere ai servizi riservati tramite un token JWT ricevuto al momento dell'autenticazione.

Il token viene generato utilizzando il metodo sign() della libreria jsonwebtoken, mentre la password viene validata tramite il metodo compareSync() della libreria bcryptjs.

```
exports.signin= (req, res) => {
    Utente.findOne({
        nome: req.body.nome,
        cognome: req.body.cognome
    })
    .populate("role", "-__v")
    .exec((err, user) => {
        if (err) {
            res.status(500).send({ message: err });
            return;
        }

        if(!user){
            return res.status(404).send({ message: "User Not found." });
        }

        if(!req.body.password){
            return res.status(400).send({message: "Nessuna password inserita"});
        }
        var passwordIsValid = bcrypt.compareSync(
            req.body.password,
            user.password
        );

        if (!passwordIsValid) {
            return res.status(401).send({
                accessToken: null,
                message: "Invalid Password!"
            });
        }

        var token = jwt.sign({ id: user.id }, process.env.SECRET, {expiresIn: 86400}); //  
24 hours

        var authority= "ROLE_" + user.role.name.toUpperCase();
    })
}
```

```
res.status(200).send({
  id: user._id,
  nome: user.nome,
  cognome: user.cognome,
  email: user.email,
  numero_telefono: user.numero_telefono,
  data_nascita: user.data_nascita,
  role: authority,
  accessToken: token
}) ;
}) ;
};
```

Aggiunta tecnico interno

Questa API permette all'amministratore di aggiungere un nuovo utente, passando le sue informazioni come body parameter. Il nuovo utente avrà il ruolo indicato del body parameter, se non viene specificato nessun ruolo il nuovo utente verrà salvato con il ruolo di tecnico interno.

Il nuovo utente viene salvato nel db tramite il metodo save().

```
exports.signup = (req, res) => {
  const utente = new Utente({
    nome: req.body.nome,
    cognome: req.body.cognome,
    email: req.body.email,
    numero_telefono: req.body.numero_telefono,
    data_nascita: req.body.data_nascita,
    password: bcrypt.hashSync(req.body.password, 8)
  });
  utente.save((err, user) => {
    if (err) {
      res.status(500).send({ message: err });
      return;
    }

    if(req.body.role){
      Role.find({ name: {$in : req.body.role}}, 
        (err, role) => {
          if(err){
            res.status(500).send({ message: err });
            return;
          }
          user.role= role.map(role => role._id);
          user.save((err => {
            if(err){
```

```
        res.status(500).send({ message: err });
        return;
    }
    res.status(201).send({ message: "User was registered successfully!" });
    return;
});;
}
);
} else{
    Role.findOne({name: "tecnico_interno"}, (err, role) => {
        if(err){
            res.status(500).send({ message: err });
            return;
        }
        utente.role=role._id;
        utente.save(err => {
            if (err) {
                res.status(500).send({ message: err });
                return;
            }

            res.status(201).send({ message: "User was registered successfully!" });
        });
    });
}
});;
```

Eliminazione tecnico interno

Questa API permette all'amministratore di eliminare un utente dal db, il quale viene selezionando passando il nome e il cognome come path parameter e trovato utilizzando il metodo `findOneAndDelete({})`. La API restituisce l'utente eliminato rispondendo con codice di stato 200.

```
const deleteUsr= (req, res) => {
    let nome= req.params.nome;
    let cognome= req.params.cognome
    Utente.findOneAndDelete({nome: nome, cognome: cognome}, (err, data) => {
        if(err) {
            return res.status(500).json({Error: err});
        }
        if(!data) {
```

```
        return res.status(404).json({message: "User '" + nome + " " + cognome + "'  
doesn't exist."});  
    }else{  
        return res.status(200).json(data);  
    }  
};  
};
```

API Documentation

Le API Locali fornite dall'applicazione ACCESS Light e descritte nella sezione precedente sono state documentate utilizzando il modulo NodeJS chiamato **Swagger UI Express**. In questo modo la documentazione relativa alle API è direttamente disponibile a chiunque veda il codice sorgente. Per poter generare l'endpoint dedicato alla presentazione delle API abbiamo utilizzato Swagger UI in quanto crea una pagina web dalle definizioni delle specifiche OpenAPI.

In particolare, di seguito mostriamo la pagina web relativa alla documentazione che presenta le API (sono presenti vari metodi GET, POST, alcuni DELETE ed un PATCH) per la gestione dei dati della nostra applicazione.

Per le API che richiedono un livello di autorizzazione è possibile eseguire il login ed utilizzare il token ricevuto inserendolo nella textbox apposita nel form di autorizzazione, al quale si accede cliccando sul pulsante “Authorization”.

L'endpoint da invocare per raggiungere la seguente documentazione e': <http://localhost:8080/api-docs>

ISS4U APIs - OpenAPI 3.0 1.0 OAS3



ISS4U Project Application API

Questa pagina contiene le APIs per utilizzare l'applicazione.

Nota che alcune APIs necessitano di una autorizzazione e quindi di un'autenticazione basata su **Bearer Token**

I sample contengono modelli di richiesta e risposta e i possibili status code associati

Matteo Parma - Andrea Gravili - Maria Laura La Face

Task

GET	/task	Prende tutte le task dal db		
POST	/task	Viene aggiunta una task		
GET	/task/{nome}	Ritorna la task con quel nome		
DELETE	/task/{nome}	Elimina la task con quel nome		
PATCH	/task/{nome}	Modifica la task selezionata		

Modulo

GET	/moduli	Ritorna la lista dei moduli		
GET	/moduli/{nome}	Ritorna il modulo con quel nome		

Utente

POST	/login	Login utente		
POST	/crew/addusr	Crea un nuovo utente		
GET	/crew	Restituisce l'elenco dell'equipaggio		
DELETE	/crew/{nome}/{cognome}	Elimina l'utente con quel nome e cognome		

Agenzie

GET	/agenzie	Restituisce l'elenco delle agenzie		
------------	----------	------------------------------------	--	--

News

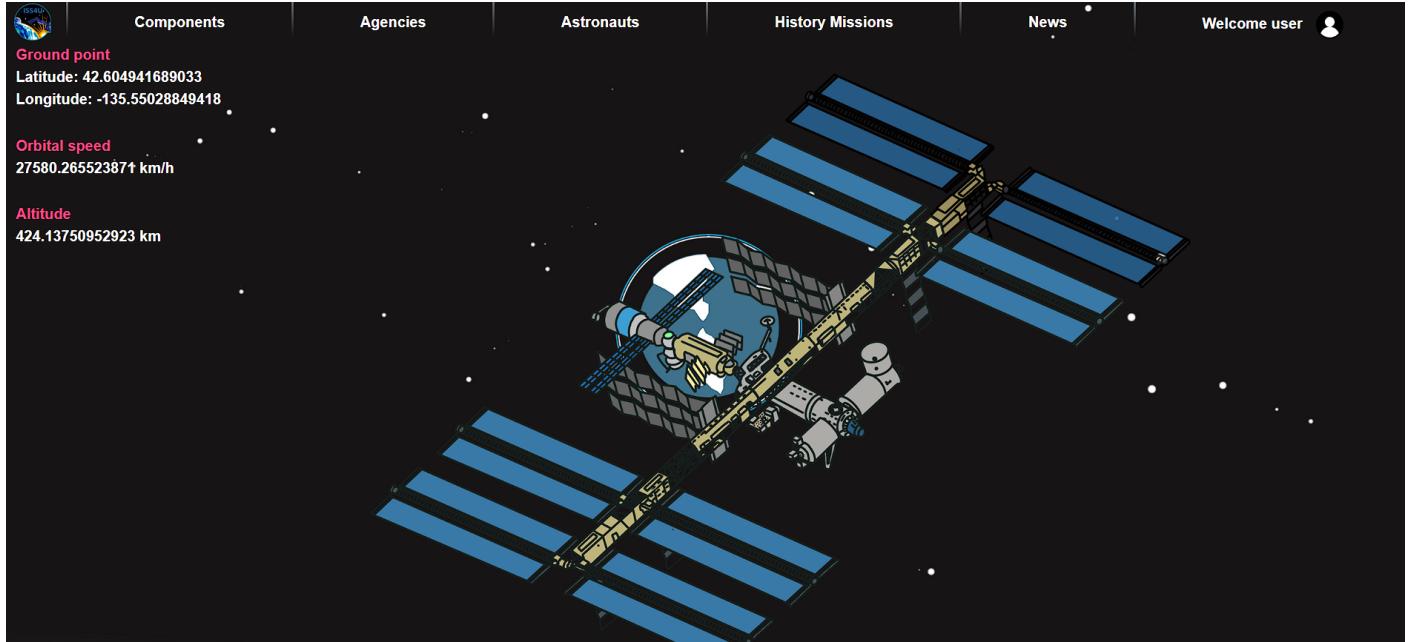
GET	/news	Restituisce tutte le news		
------------	-------	---------------------------	--	--

FrontEnd Implementation

Il front-end fornisce le funzionalità di visualizzazione delle varie pagine, informazioni riguardanti tutti i dati di iss4u, tra cui: componenti della iss, news, agenzie, astronauti, storia delle missioni, news.

In particolare permette il display delle informazioni principali di esse, che vengono prelevate tramite il back-end, inoltre il front-end permette il login, il logout, e a seconda dell'accesso permette di utilizzare determinate funzionalità, come la gestione degli utenti che può essere effettuata solo dagli amministratori.

Home page - componenti



Questa è la home page, dove in alto avremo la barra di navigazione per ogni pagina, che sarà presente in ogni pagina navigata. Cliccando su “welcome user” potremo accedere al login, mentre le altre pagine daranno accesso a quello che indicano.

Nella home page, se scorreremo verso il basso, troveremo i componenti della iss:

Cerca componente... (nome o descrizione)

COMPONENTI



Unity



Zarya

ISS4U

ISS4U è un'applicazione creata con lo scopo di rendere più semplice il concetto di ISS e cosa realmente c'è dietro per chiunque. Ma non solo, ISS4U è anche un gestionale per chi dovrebbe essere a bordo e chi a terra

LINK UTILI

[Nasa](#)
[Esa](#)
[Iss](#)

GITHUB

[ISS4U](#)
[Andrea Gravili](#)
[Maria Laura La Face](#)
[Matteo Parma](#)

Documento: ISS4U_Sviluppo-app

Delivery: 04

Possiamo anche attraverso una barra di ricerca poter cercare dal nome o dalla descrizione un componente, inoltre se passiamo con il mouse sopra ad un componente potremo avere una breve descrizione di cosa si tratta:



Unity

Il modulo Unity è stato il primo ad essere collegato alla ISS. Serve come hub di collegamento tra i vari moduli e come camera di equilibrio.

E al clic sul componente potremo avere una visualizzazione di ogni dato:

The screenshot shows a 3D model of the International Space Station's Unity module. A search bar at the top left contains the text "Cerca componente... (nome o descrizione)". On the right, a detailed information card is displayed for the "Unity" module:

Unity

Descrizione: Il modulo Unity è stato il primo ad essere collegato alla ISS. Serve come hub di collegamento tra i vari moduli e come camera di equilibrio.

Nazione: Stati Uniti

Tipo / Applicazione: Abitazione e supporto tecnico

Operatore: NASA

Contratti: Boeing

Energia: 8 kW

Massa: 19.7 t

Launch date: 4 dicembre 1998

Altri dettagli: Il modulo Unity è stato lanciato il 4 dicembre 1998 a bordo del shuttle orbitale Endeavour

Matteo Parma

In the bottom left corner of the main window, there is a small "ISS4U" logo with descriptive text: "ISS4U è un'applicazione che permette di conoscere in modo semplice il concetto di ISS. Permette di scoprire chi a bordo, chi a terra gestionale per chi dovrebbe essere a bordo e chi a terra".

In fondo ad ogni pagina possiamo trovare il footer con informazioni aggiuntive sull'applicazione.

Agenzie

Nella pagina delle agenzie potremo trovare ogni agenzia che ha contribuito in qualche modo all'evoluzione della ISS, con ogni logo che le rappresenta e se ci si clicca sopra si viene reindirizzati alla pagina del sito. (causa guerra in Ucraina la pagina dell'ente spaziale russo **roskosmos** non è disponibile).

AGENZIE



Astronauti

La pagina degli astronauti si apre subito con uno slider iniziale che può essere tranquillamente fatto scorrere dall'utente.



Subito sotto troviamo gli astronauti che sono realmente i tecnici interni / amministratori che sono registrati attualmente sulla ISS. In caso di mancanza di foto viene messo un placeholder. Anche in questo caso possiamo cercare gli astronauti a nostra scelta tramite una barra di ricerca apposita.

Cerca componente... (nome o descrizione)

ASTRONAUTI



**Samantha
Cristoforetti**

Nascita: 26-04-1977

Occupazione: comandante

Missioni: Expedition 42/43 - Missione Minerva - Expedition 64

Biografia: Samantha Cristoforetti è un'astronauta e aviatrice italiana, prima donna italiana negli equipaggi dell'Agenzia Spaziale Europea e prima donna europea comandante della Stazione spaziale internazionale.



Jessica Watkins

Nascita: 14-05-1988

Occupazione: Mission specialist

Missioni: Expedition 64

Biografia: Jessica Andrea Watkins è un'astronauta e geologa statunitense. Nel 2022 ha preso parte alla missione di lunga durata SpaceX Crew-4 (Expedition 67) a bordo della Stazione spaziale internazionale

History mission

HISTORY MISSION



Expedition 1

La prima Expedition segnò l'occupazione continua dell'ISS da parte dell'equipaggio umano. Durante questa missione, l'equipaggio svolse una serie di esperimenti scientifici e test di sistemi vitali.

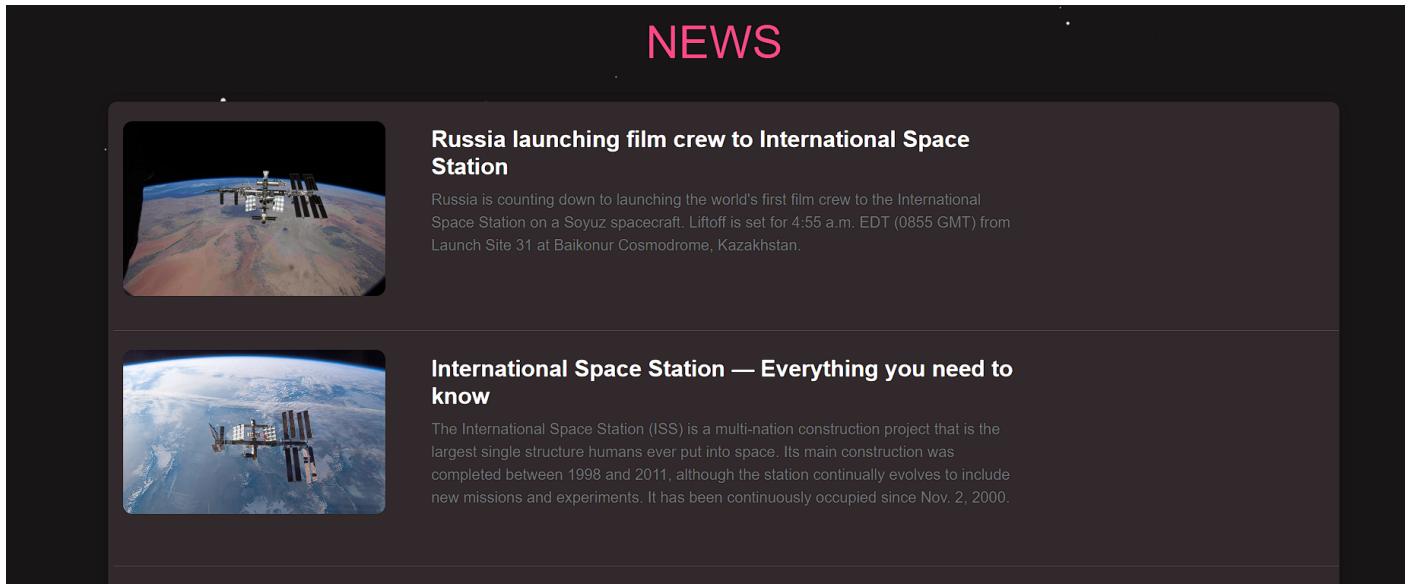


Expedition 2

L'equipaggio di Expedition 2 continuò la ricerca scientifica e i test di sistemi cruciali per il funzionamento dell'ISS. La missione contribuì a stabilire la presenza costante dell'uomo nello spazio.

Nella history mission troviamo tutte le spedizioni effettuate, con un'immagine sulla sinistra che rappresenta la spedizione e alla sua destra il titolo e la descrizione di essa subito sotto

News



Le news sono mostrate nello stesso layout della pagine delle history mission.

Login

Il login è mostrato come un form centrale allo schermo con le opzioni di mettere il proprio nome, cognome e password ed una eventuale gestione degli errori in caso di mancanza di dati inseriti

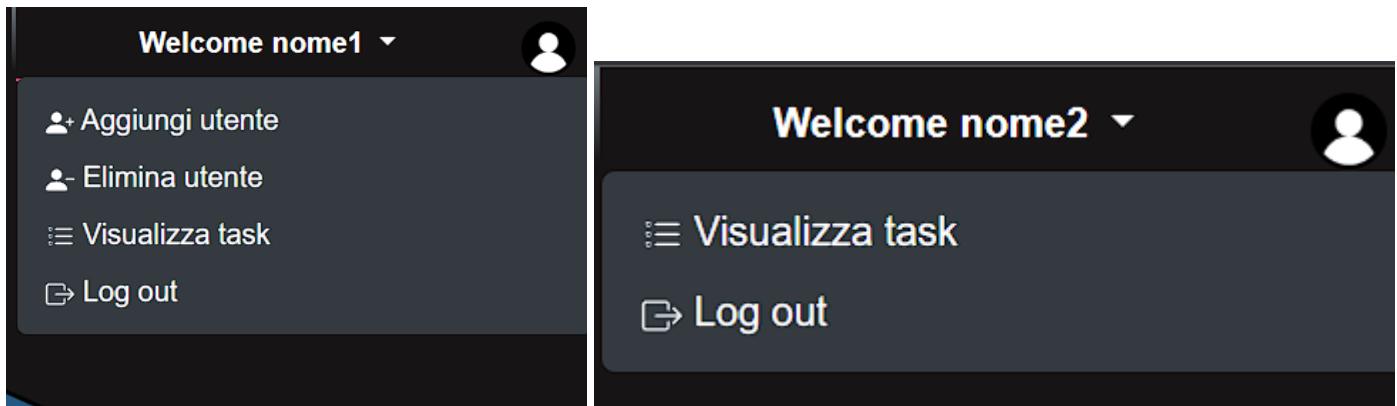


In caso di corretto inserimento dei dati, si viene riportati sulla home e verrà mostrato sulla barra di ricerca che si è correttamente loggati:



Inoltre, dopo il log, se clicchiamo sulla colonna della barra di navigazione di benvenuto, non verremo più portati alla pagina di login come prima, ma si aprirà un menù a tendina, il primo indica un menu a tendina

che compare che un amministratore ha fatto l'accesso, mentre il secondo è se un tecnico interno ha fatto l'accesso.



Nel menu a tendina il log-out in realtà non porta a nessuna pagina, ma effettua tutte le operazioni per sloggarsi e ritorna alla home page.

Visualizza task

Nella pagina di visualizzazione delle task ci apparirà per prima cosa una lista di tutte le task attualmente attive, se è un amministratore a vederle allora vorrà dire che verranno visualizzate tutte le task, altrimenti se è un tecnico interno allora verranno visualizzate solamente le sue.

Qui le task si potranno solo scorrere, inoltre ci sarà una barra di ricerca apposita per ricercare una task cercando direttamente dal nome.

Le task sono visualizzate in modo tale che alla sinistra avremo il numero della task, il suo titolo in grassetto e il codice dell'utente a cui è associata. Sulla destra invece la data di inizio e di fine della task.

Subito sotto il titolo avremo il modulo a cui è destinato e sotto ancora la descrizione della task.

The screenshot shows a task management application interface. At the top, there is a search bar with the placeholder "Cerca task... (nome)" and a magnifying glass icon. Below the search bar is a list of tasks:

- 6. Manutenzione Modulo** - 63eb60a8b84d4069cf623747
Unity
Verifica e sostituzione di componenti hardware e software all'interno del Modulo Unity per garantire il corretto funzionamento dei sistemi di comunicazione e controllo. Ispezione dei pannelli solari e riparazione di eventuali danni.
2023-09-11T00:00:00.000Z - 2023-09-16T00:00:00.000Z
- 7. Installazione Nuovi Esperimenti** - 64439d3111712c892233d6fc
Unity
Installazione di nuovi apparati e strumenti scientifici per condurre esperimenti in microgravità. Configurazione e verifica del corretto funzionamento degli strumenti prima dell'avvio degli esperimenti.
2023-07-22T23:01:12.102Z - 2023-07-30T23:01:12.102Z
- 8. Aggiornamento Sistemi di Supporto Vitale** - 64bc12ad20e8ac7563eaa683
Destiny
Aggiornamento dei sistemi di supporto vitale per garantire la sicurezza e il funzionamento corretto dei servizi essenziali.
2023-11-03T23:01:12.102Z - 2023-11-10T23:01:12.102Z

Subito sotto la lista delle task troveremo tre buttoni con tre diverse opzioni:



Aggiunta task

Cliccando il bottone per l'aggiunta delle task, ci si aprirà un form per l'aggiunta di una task con tutti i dati appropriati, inoltre con rilevazione degli errori in caso che ci siano.

Aggiungi task

Data di inizio
gg/mm/aaaa

Data di fine
gg/mm/aaaa

Nome task

Modulo

Descrizione

Completato?

Selezione un utente

Aggiungi

Il menu di selezione dell'utente è presente solamente per gli amministratori e per i tecnici interni non comparirà.

Aggiungi task

409: Esiste già una task con questo nome

Data di inizio
28/08/2023

Data di fine
28/08/2023

Nome task
prova

Modulo

(esempio di errore durante l'aggiunta)

Eliminazione task

Cliccando l'eliminazione delle task troveremo un nome di una task da eliminare, con eventuale rilevazione degli errori.

Elimina task

Nome task

Sei sicuro di voler eliminare questa task?

Elimina

Elimina task

Devi spuntare la casella per eliminare la task

Nome task
prova

Sei sicuro di voler eliminare questa task?

Elimina

Modifica task

Cliccando modifica task inizialmente ci comparirà solamente un menù a tendina con all'interno ogni task presente nel database:

Modifica task

Selezione task:

- Selezione una task
- Manutenzione Modulo
- Installazione Nuovi Esperimenti
- Aggiornamento Sistemi di Supporto Vitale**
- Riparazione Struttura Esterna
- Ottimizzazione Sistema Energia Solare

Documento: ISS4U_Sviluppo-app

Delivery: 04

Selezionando una di queste task ci si aprirà un intero form precompilato con i dati della task e che potremo direttamente modificare tramite i campi di input

Modifica task

Selezione task:

Manutenzione Modulo

Nome task da modificare
Manutenzione Modulo

Data di inizio
11/09/2023 

Data di fine
16/09/2023 

Modulo
Unity

Descrizione Verifica e sostituzione di componenti hardware e software all'interno del Modulo Unity per garantire il corretto funzionamento dei sistemi di 

Completato?

Modifica

Aggiunta utente

L'aggiunta utente è un'opzione solamente per gli utenti loggati come amministratori, ci comparirà un form che potremo compilare al fine di aggiungere un utente al database, dove quando verrà inserito correttamente, verremo reindirizzati alla pagina degli astronauti al fine di vedere correttamente l'aggiunta effettuata.

AGGIUNGI UTENTE

Nome

Cognome

@ E-mail

Telefono +39

Data di nascita gg/mm/aaaa

Password

Ruolo: tecnico interno

Bio

Occupazione

Missioni

Elimina utente

La elimina utente conterrà solamente un form dove si dovrà inserire il nome e il cognome di un utente presente al fine di eliminarlo. Anche questa sezione è accessibile solo da utenti loggati come amministratori.

ELIMINA UTENTE

Nome

Cognome

Sei sicuro di voler eliminare questo utente?

GitHub Repository & Deployment Info

La repository git è disponibile al seguente link: [github](#). I contenuti dell'applicazione sono disponibili nelle repositories **back-end**, **front-end**. All'interno di esse è presente il codice sviluppato per le rispettive parti dell'applicazione.

Per utilizzare le funzionalità degli utenti registrati è possibile accedere come amministratore con le seguenti credenziali:

- nome: Samantha
- cognome: Cristoforetti
- password: 12345678

oppure è possibile accedere come tecnico interno con:

- nome: Jessica
- cognome: Watkins
- password: 12345678

L'applicazione si troverà al seguente link: [ISS4U](#).

Per far girare l'applicazione in locale si dovrà clonare la repository *back-end*. Dopo di che va inserito un file .env con le seguenti variabili d'ambiente:

MONGOATLASURI=“mongodb+srv://T28:T28_pass@cluster0.y64v6lv.mongodb.net/?retryWrites=true&w=majority”

PORT=8080

SECRET=“ISS4U-secret-key”

Dopo di che eseguire **npm install && npm start** e la web app sarà attiva all'indirizzo <http://localhost:8080>

Testing

Il testing è stato fatto utilizzando Jest, node-fetch e supertest.

Nel testing delle task i casi di test sono stati fatti sempre sia con l'accesso amministratore che con il livello di autorizzazione di un tecnico interno.

```
Test Suites: 6 passed, 6 total
Tests:       21 passed, 21 total
Snapshots:   0 total
Time:        9.401 s
Ran all test suites.
```

Tutti i casi di test sono elencati nella seguente tabella.

Numero test case	Descrizione test case	Test data	Precondizioni	Risultato atteso	Risultato riscontrato
1	Login	<nome> <cognome> <password>	L'utente è registrato e le credenziali sono corrette	L'utente accede come tecnico interno o amministratore	Analogo al risultato atteso
1.1	Login di un utente non registrato o con credenziali errate	<nome> <cognome> <password>	L'utente non possiede un account	Messaggio "user not found" con status code 404 In caso di solo password errata messaggio "invalid password" (401)	Analogo al risultato atteso
1.2	Login in mancanza di una o più credenziali	<nome> <cognome> <password> <i>presenti o non</i>		Messaggio di mancanza di credenziali	Analogo al risultato atteso
2	Aggiunta task	<nome> <data fine> <data inizio> <modulo> <descrizione> (opzionale) <completata> <nameuser> (se admin) <cognomeuser> (se admin)	sono presenti tutti gli attributi e non esiste già una task con quel nome	Aggiunta nella task nel db con codice 201 e con <userId> corretto	Analogo al risultato atteso
2.1	Aggiunta task con lo stesso nome	<nome> <data fine> <data inizio> <modulo> <descrizione> (opzionale) <completata> <nameuser> (se admin) <cognomeuser> (se admin)		Il sistema avverte che esiste già una task con quel nome con codice di stato 409 e non salva la task nel db	Analogo al risultato atteso
2.2	Aggiunta task con riferimento ad un utente non esistente	<nome> <data fine> <data inizio> <modulo> <descrizione> (opzionale) <completata> <nameuser> <cognomeuser>	l'utente <nameuser><cognomeuser> non è presente nel db	Il sistema risponde con codice 404 e con messaggio "utente non trovato"	Analogo al risultato atteso
2.3	Aggiunta task con attributi required mancanti	Almeno uno di questi mancate <nome> <data fine> <data inizio>		Messaggio di mancanza di attributi	Analogo al risultato atteso

3	Modifica della task	<nomeoriginale> <nome> <data fine> <data inizio> <modulo> <descrizione> <completata>	la task è già presente nel db	La task viene modificata con successo con codice 200	Analogo al risultato atteso
3.1	Modifica di una task non esistente o non appartenente all'utente	<nomeoriginale> <nome> <data fine> <data inizio> <modulo> <descrizione> <completata>		Messaggio di errore dicendo che la task non è presente o che non appartiene all'utente. Codice 404	Analogo al risultato atteso
4	Eliminazione task	<nome>	La task è presente nel db	Eliminazione con successo (200)	Analogo al risultato atteso
4.1	Eliminazione task non esistente	<nome>		Messaggio di errore dicendo che non è presente nessuna task con quel nome (404)	Analogo al risultato atteso
4.2	Eliminazione task non appartenente a quel tecnico interno		L'utente che intende eliminare la task è registrato come tecnico interno	Messaggio con errore dicendo che la task non è stata trovata	Analogo al risultato atteso
5	Lista task - admin		L'utente ha fatto l'accesso come amministratore	Ottiene la lista di tutte le task di tutti i tecnici interni con codice di stato 200	Analogo al risultato atteso
5.1	Lista task - tecnico interno		L'utente ha fatto l'accesso come tecnico interno	Ottiene la lista delle task appartenenti a quell'utente con codice di stato 200	Analogo al risultato atteso
6	Aggiunta utente	<nome> (required) <cognome> (req.) <email> <password> (req.) <role> <data di nascita> <numero di telefono> <bio> <occupazione> <missioni>	Sono presenti almeno gli attributi required e non è già presente un utente con stesso nome e cognome	Aggiunte dell'utente con successo	Analogo al risultato atteso
6.1	Aggiunta utente con nome e cognome già presenti	<nome> (required) <cognome> (req.) <email> <password> (req.) <role> <data di nascita> <numero di		Messaggio di errore dicendo che è già presente un utente con quel nome e cognome	Analogo al risultato atteso

		telefono> <bio> <occupazione> <missioni>			
7	Eliminazione utente	<nome> <cognome>	L'utente è registrato	Eliminazione con successo	Analogo al risultato atteso
7.1	Eliminazione utente non esistente	<nome> <cognome>		Messaggio di errore dicendo che l'utente non non esiste (404)	Analogo al risultato atteso
8	Lista moduli			Ottiene la lista dei moduli nel db (200)	Analogo al risultato atteso
9	Lista crew			Ottiene la lista degli utenti registrati (200)	Analogo al risultato atteso
10	Lista agenzie			Ottiene la lista delle agenzie nel db	Analogo al risultato atteso